Universidad Nacional de Loja

FACULTAD DE LA ENERGÍA, LAS INDUSTRIAS Y LOS RECURSOS NATURALES NO RENOVABLES

CARRERA DE INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

TERCER CICLO “A”

ASIGNATURA: Estructura de Datos

ELABORADO POR:

Anderson Mateo Coello Jaramillo

Mark Anthony Gonzalez Jaramillo

Docente encargado: ING. Andres Navas

FECHA DE ENTREGA: 21/11/2025

LOJA – ECUADOR

- **Discusión entre integrantes**
  En este trabajo comparamos los tres algoritmos que vimos en clase: Burbuja, Selección e Inserción, aplicándolos a diferentes tipos de datos. En general, cada algoritmo funciona mejor dependiendo de cómo están organizados los datos antes de ordenar.

  Burbuja es bueno cuando los datos ya están casi ordenados, porque se da cuenta rápido y termina antes. Pero cuando los datos están al revés, Burbuja se vuelve uno de los más lentos porque necesita hacer muchos intercambios.

  Selección es el algoritmo más "parejo". Siempre tarda más o menos lo mismo sin importar el orden de los datos. Su ventaja es que hace muy pocos intercambios, así que es bueno si se quiere mover la menor cantidad de elementos posible.

  La inserción es más rápida cuando los datos tienen cierto orden inicial, especialmente si están casi ordenados. Pero cuando están totalmente invertidos, la Inserción tiene bastantes movimientos y se vuelve más lento.

- **Matriz de Resultados por dataset**

  1. Dataset: citas_100.csv
  n: 100.

  tipo de dataset: aleatorio.

n	tipo de dataset	algoritmo	comparisons	swaps	tiempo_mediana (ns)
100	**aleatorio**	**Burbuja**	**4950**	**2525**	**290,000**
100	**aleatorio**	**Selección**	**4950**	**87**	**200,000**
100	**aleatorio**	**Inserción**	**2530**	**2434**	**200,000**

2. Dataset: citas_100_casi_ordenadas.csv

n: 100.

tipo de dataset: casi-ordenado.

n	tipo de dataset	algoritmo	comparisons	swaps	tiempo_mediana (ns)
100	casi-ordenado	Burbuja	4949	459	790,000
100	casi-ordenado	Selección	4950	87	200,000
100	casi-ordenado	Inserción	558	459	110,000

3. Dataset: pacientes_500.csv

n: 500

Tipo de dataset: duplicados (o aleatorio con muchos duplicados)

n	tipo de dataset	algoritmo	comparisons	swaps	tiempo_mediana (ns)
500	duplicados	Burbuja	124597	67719	3,540,000
500	duplicados	Selección	124750	492	4,160,000
500	duplicados	Inserción	68213	67719	2,680,000

4. Dataset: inventario_500_inverso.csv

n: 500

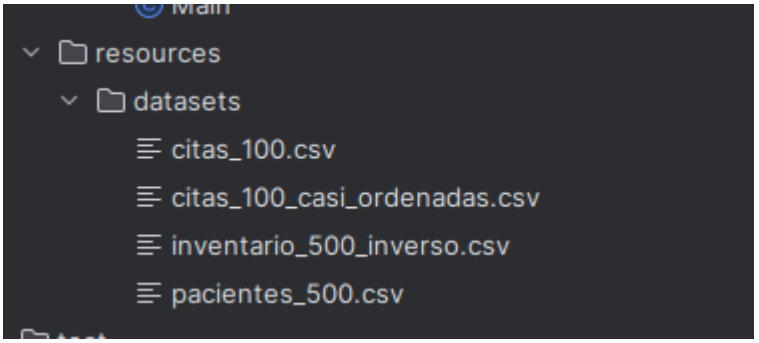
tipo de dataset: inverso (orden estrictamente descendente).

n	tipo de dataset	algoritmo	comparisons	swaps	tiempo_mediana (ns)
500	inverso	Burbuja	123622	61564	4,140,000
500	inverso	Selección	124750	494	2,710,000
500	inverso	Inserción	62058	61564	3,380,000

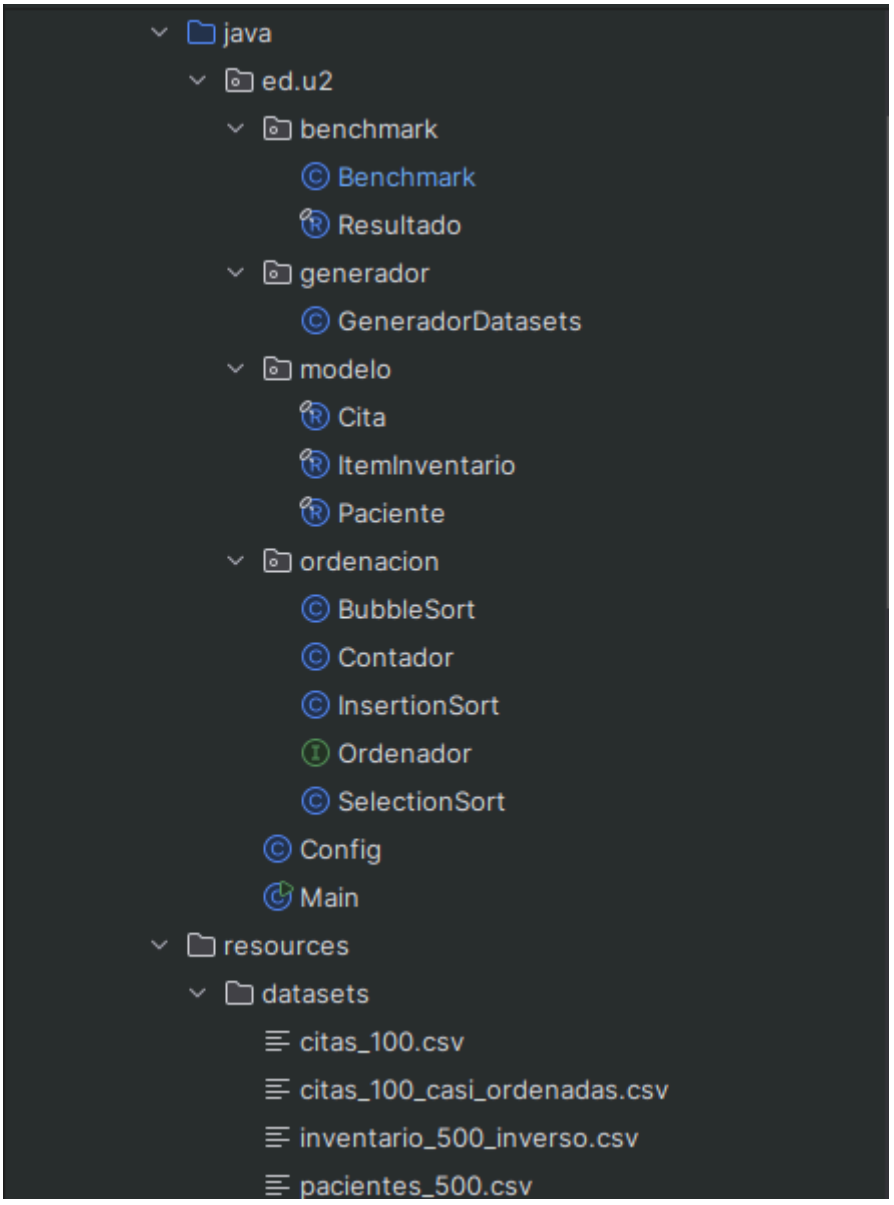
- **Matriz de Recomendación:**

Algoritmo	Dataset Aleatorio	Dataset Casi Ordenado	Dataset Inverso	Recomendado
Burbuja	Rendimiento bajo. Muchas comparaciones y swaps.	Rinde mucho mejor porque detecta orden y puede cortar temprano.	Extremadamente lento.	Solo se usa en listas muy pequeñas o casi ordenadas.
Selección	Rendimiento estable, siempre hace las mismas comparaciones.	Rinde mucho mejor porque detecta orden y puede cortar temprano.	Obtiene el mismo desempeño.	Solo se usa en listas muy pequeñas o casi ordenadas.
Inserción	Bueno para tamaños pequeños y medianos.	Muy eficiente cuando la lista está casi ordenada.	Muy lento en inversos porque desplaza muchos elementos.	Recomendado cuando se espera orden parcial. Ideal para datos “casi ordenados”.

- **Los 4 datasets en formato CSV:**



- **Código fuente de Java:**



```java
package ed.u2.generador;

import ed.u2.Config;
import ed.u2.modelo.Cita;
import ed.u2.modelo.Paciente;
import ed.u2.modelo.ItemInventario;

import java.io.FileWriter;
import java.time.LocalDateTime;
import java.util.*;

public class GeneradorDatasets {  3 usages  AndersonLoop

    private final Random random = new Random(Config.SEMILLA);  8 usages

    public void generarTodos() throws Exception {  1 usage  AndersonLoop
        generarCitas100();
        generarCitas100CasiOrdenadas();
        generarPacientes500();
        generarInventario500Inverso();
    }

    private void generarCitas100() throws Exception {  1 usage  AndersonLoop
        List<Cita> lista = new ArrayList<>();

        for (int i = 1; i <= 100; i++) {
            String id = "C" + i;
```

```java
package ed.u2;

public class Config {  15 usages  AndersonLoop

    public static final long SEMILLA = 42L;  1 usage

    public static final String RUTA = "src/main/resources/datasets/";  4 u

    public static final String C100 = "citas_100.csv";  2 usages
    public static final String C100CO = "citas_100_casi_ordenadas.csv";
    public static final String P500 = "pacientes_500.csv";  2 usages
    public static final String I500 = "inventario_500_inverso.csv";  2 usa
}
```

```java
package ed.u2.benchmark;

import ...

public class Benchmark {  3 usages  AndersonLoop +1 *

    public void ejecutarPruebas() throws Exception {  1 usage  AndersonLoop

        probarDataset(Config.C100, this::cargarCitas, Comparator.comparing(Cita::apellido));
        probarDataset(Config.C100CO, this::cargarCitas, Comparator.comparing(Cita::apellido));
        probarDataset(Config.P500, this::cargarPacientes, Comparator.comparing(Paciente::priori
        probarDataset(Config.I500, this::cargarInventario, Comparator.comparing(ItemInventario:
    }

    private <T> void probarDataset(  4 usages  AndersonLoop
            String nombreArchivo,
            Cargador<T> cargador,
            Comparator<T> comparador
    ) throws Exception {

        System.out.println("\n========================================================");
        System.out.println(" DATASET: " + nombreArchivo);
        System.out.println("========================================================");

        List<Resultado> resultados = new ArrayList<>();

        resultados.add(medir( nombreAlgoritmo: "Burbuja", new BubbleSort<>(), cargador, nombreArchi
```

```java
package ed.u2.ordenacion;

import java.util.Comparator;
import java.util.List;

public interface Ordenador<T> {  4 usages  3 implementations  AndersonLoop
    void ordenar(List<T> lista, Comparator<T> comparador, Contador contador);  1
}
```

- Capturas de ejecución:

```
========================================================
 DATASET: citas_100.csv
========================================================
-------------------------------------------------------------
| Algoritmo    | Tiempo (ms) | Comparaciones  | Swaps      |
-------------------------------------------------------------
| Burbuja      |        0.02 |             99 |          0 |
| Selección    |        0.63 |           4950 |          0 |
| Inserción    |        0.02 |             99 |          0 |
-------------------------------------------------------------
```

```
======================================================
 DATASET: citas_100_casi_ordenadas.csv
======================================================
-------------------------------------------------------------
| Algoritmo    | Tiempo (ms) | Comparaciones  | Swaps      |
-------------------------------------------------------------
| Burbuja      |        0.79 |           4949 |        459 |
| Selección    |        0.20 |           4950 |         87 |
| Inserción    |        0.11 |            558 |        459 |
-------------------------------------------------------------
```

```
======================================================
 DATASET: pacientes_500.csv
======================================================
-------------------------------------------------------------
| Algoritmo    | Tiempo (ms) | Comparaciones  | Swaps      |
-------------------------------------------------------------
| Burbuja      |        3.54 |         124597 |      67719 |
| Selección    |        4.16 |         124750 |        492 |
| Inserción    |        2.68 |          68213 |      67719 |
-------------------------------------------------------------
```

```
======================================================
 DATASET: inventario_500_inverso.csv
======================================================
-------------------------------------------------------------
| Algoritmo    | Tiempo (ms) | Comparaciones  | Swaps      |
-------------------------------------------------------------
| Burbuja      |        4.14 |         123622 |      61564 |
| Selección    |        2.71 |         124750 |        494 |
| Inserción    |        3.38 |          62058 |      61564 |
-------------------------------------------------------------
```

Preguntas de Control:

1. ¿Por qué imprimir trazas durante la medición distorsiona los tiempos?

- Porque las operaciones de E/S (Input/Output) como System.out.println son muy lentas comparadas con las operaciones en memoria.
- El tiempo medido incluiría el overhead de imprimir en consola/archivo, en lugar de reflejar sólo el tiempo real del algoritmo.
- El recolector de basura y la optimización JIT también pueden verse afectados por estas operaciones externas.

2. Explica por qué Selección tiene comparaciones ~ n (n–1)/2 sin importar el orden inicial.

- Porque el algoritmo de Selección siempre recorre todo el segmento no ordenado del array para encontrar el mínimo/máximo.
- Realiza todas las comparaciones posibles entre elementos, sin importar si el array ya está ordenado o no.
- La fórmula n(n-1)/2 corresponde al número total de comparaciones en el peor, mejor y caso promedio.

3. ¿Por qué la Inserción es competitiva en datos casi ordenados?

- Porque en datos casi ordenados, el algoritmo de Inserción realiza muy pocos desplazamientos.
- Solo necesita hacer comparaciones y movimientos locales cerca de la posición actual.
- En el mejor caso, su complejidad es O(n), mucho mejor que los otros dos algoritmos.

4. ¿Qué papel juegan los duplicados en la estabilidad del resultado?

- Mantienen su orden relativo original cuando se usa un algoritmo estable.
- Inserción y Burbuja son estables, por lo que preservan el orden de duplicados.
- Selección es inestable por defecto, ya que intercambia elementos distantes, lo que puede alterar el orden de duplicados.

5. ¿Por qué Burbuja con corte temprano mejora en “casi ordenado” pero no en “inverso”?

- En "casi ordenado", Burbuja con corte temprano detiene la ejecución rápidamente al no encontrar intercambios.
- En "inverso", el array está totalmente desordenado, por lo que el algoritmo siempre realiza todos los pases y comparaciones posibles.
- El "corte temprano" solo ayuda cuando el array está cerca de estar ordenado, no cuando está invertido.